姓名：颜学海

支付宝账号：yxh2018@yahoo.com.cn

12:00 - 13:00 每一个人，需要将自己的考勤，12次课，考勤，包括每次作业情况，做成一个文档，交给班长，造假，分数不达标

作业在12:00之前完毕并统统一交给班长，班长统一压缩在12:05之前发到微信群

16:00 - 18:00 下午16:00 - 18:00 老师将随机抽取学员回答两道问题，进行答辩，并最终统计学分，按照规则，达90分学员，财务周一将将其公链学费的80%返还到学员支付宝。

1. 尽可能多的写出整个课程，我们学习并掌握到的知识点

通过学习go公链开发，掌握了go编程，掌握了区块链的相关理论和编程实战

1. Github fork及提交项目及修改
2. 区块的概念和结构，区块链的概念和结构，如何产生创世区块及新区块，区块链
3. Pow挖矿原理入编程实现，工作量证明的算法，pos及dpos的原理
4. 区块hash的实现及编程。区块hash的产生及验证有效性
5. 区块的序列化和反列化，boltdb的安装及应用，库，表的创建，数据的增删改查
6. CLI命令行命令及参数的调用和编程实现
7. UTXO模型，余额查询及转帐交易处理，跌代器遍历
8. 加密及签名，随圆曲线加密，Base58
9. 钱包的结构及地址生成，私钥公钥
10. 将区块及钱包生成相关的文件 保存到数据库文件
11. 钱包节点，全节点，矿工节点的部署及实现，区块数据同步
12. P2p协议及一致性算法的实现，otto,etcd环境搭建，原理及实现
13. 虚拟机，智能合约的编程，部署及实现调用

2. 描述我们课程中POW代码实现逻辑

一、生成区块的原始数据

二、将前一区块的hash,交易数据，时间戳，目标难度，自增的nonce值，区块高度生成区块hash

三、如果产生的hash值没有达到指定的难度，则nonce值加1,再次产生hash，直到完成难度系统的要求为止

四、将这个hash及nonce值传给block, 组织数据后进行交易验证，签名及验证，并保存到区块链中和数据库中

3. 课程中数据库如何实现增删改查

Bolt就是这么一个纯粹的Go语言版的嵌入式key/value的数据库，而且在Go的应用中很方便地去用作持久化

一、打开数据库,没有就需要创建db, err := bolt.Open("my.db", 0600, &bolt.Options{Timeout: 1 \* time.Second})，注意后面带上defer db.clode()

二、创建表，table,err = tx.CreateBucket([]byte("block"))

三、增加db.update：以key,value形式存储table.Put([]byte("yxh135279"),blockBytes)

四、修改db.update：以key,value形式修改table.Put([]byte("yxh135279"),blockBytes)

五、删除db.update：table.Delete(([]byte("yxh135279"))

六、查询db.update：blockBytes := table.Get([]byte("yxh135279"))

4. 图文并貌完整的描述钱包地址生成过程

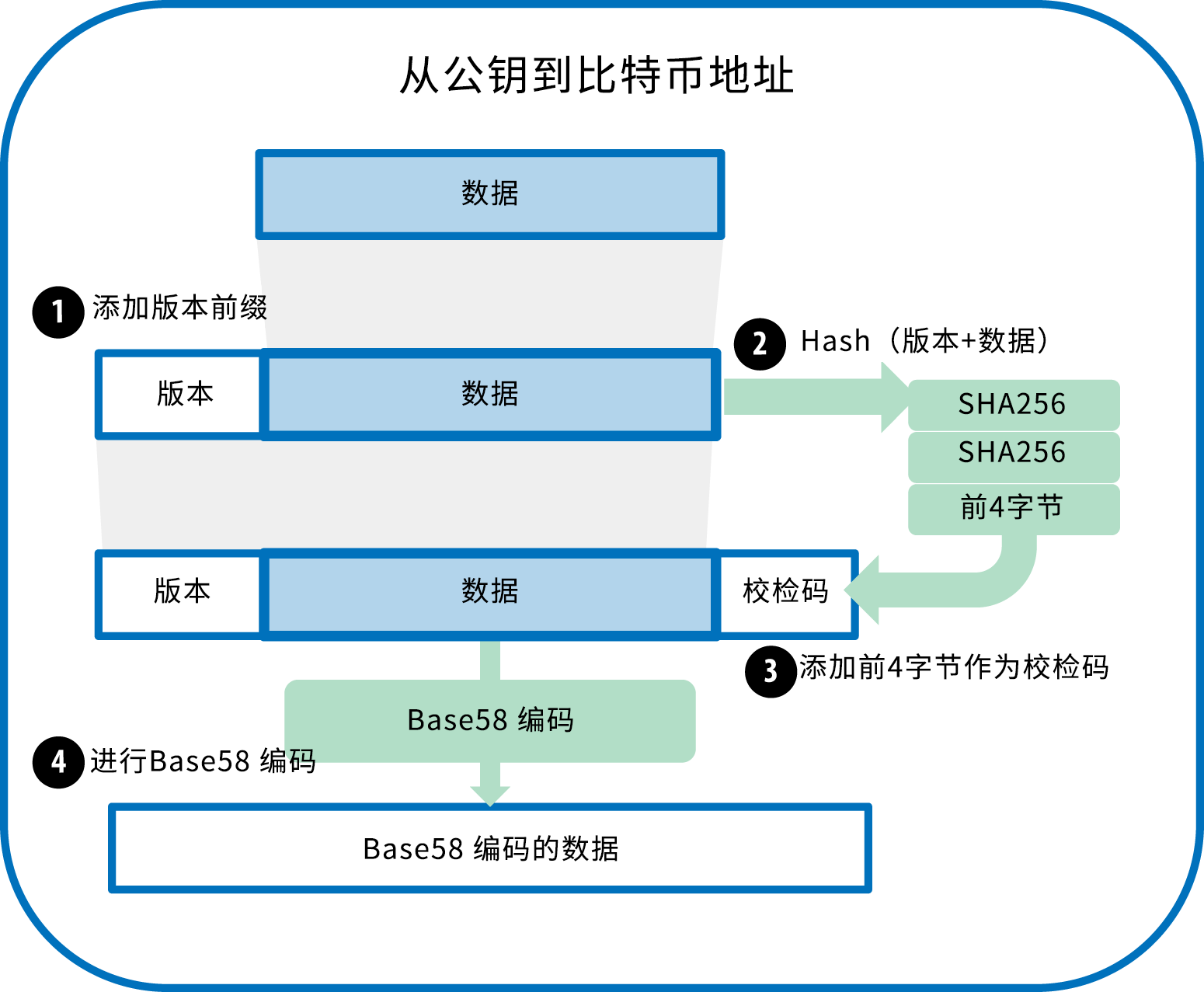
一、生成一个钱包集wallets存放钱包map的key用钱包的地址，value对应每个钱包数据

二、根据ecdsa.PrivateKey曲线生成公私钥对

三、公钥通过sha256和ripemd160两次加密，产生公钥哈希

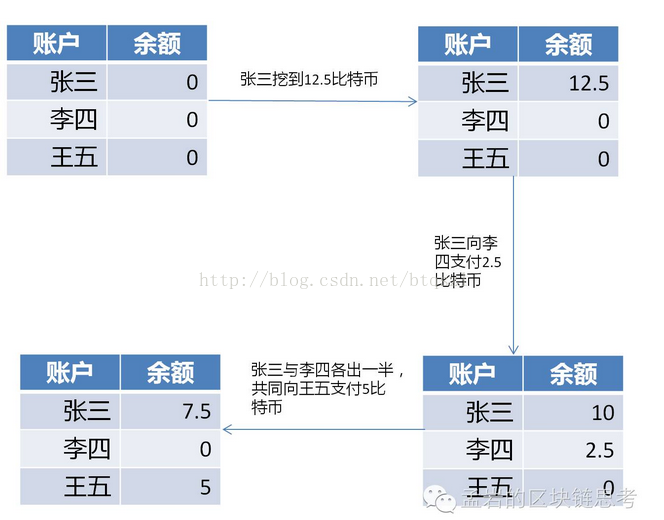
四、将版本+公钥hash通过两次sha256加密，截取前面4位为验证码

五、将版本，版本+公钥hash通过两次sha256加密，截取前面4位为验证码组合再通过Base58最终形成钱包地址(如果第一位不是1，则需要替换成1)



5. 图文并貌描述据两个实例描述UTXO模型的巧妙设计

张三挖到12.5 枚比特币。过了几天，他把其中 2.5 枚支付给李四。又过了几天，他和李四各出资 2.5 比特币凑成 5 比特币付给王五。  
如果是基于账户的设计，张、李、王三人在数据库中各有一个账户，则他们三人的账户变化如下图所示:





每笔交易都有若干交易输入，也就是资金来源，也都有若干笔交易输出，也就是资金去向。一般来说，每一笔交易都要花费（spend）一笔输入，产生一笔输出，而其所产生的输出，就是“未花费过的交易输出”，也就是 UTXO。  
比特币交易遵守几个规则。  
第一，除了 coinbase 交易之外，所有的资金来源都必须来自前面某一个或者几个交易的 UTXO，就像接水管一样，一个接一个，此出彼入，此入彼出，生生不息，钱就在交易之间流动起来了。  
第二，任何一笔交易的交易输入总量必须等于交易输出总量，等式两边必须配平。

6. 私钥签名，公钥是如何验证签名的

一、通过随机数，私钥，交易数据用曲线计算方式进行签名

二、计算签名的长度，对半分成2个字节数组r,s

三、通过公钥hash分成2半,x,y，通过ecdsa.PublicKey，curve,x,y计算rawPublic

四、通过ecdsa.Verify(&rawPublic,交易数据, &r,&s)进行比较，相同则通过验证

7. 完整的描述节点区块同步的逻辑和过程

一、各节点设置好NODE\_ID

二、启动主节点

三、启动钱包节点，启动后向主节点发送自己的版本，区块高度，地址信息version

四、主节点收到后对比，将自身的高度信息发送给钱包节点version

五、钱包节点对比高度，如果小于主节点的高度，则请求区块hash列表blocks

六、主节点收到列表请求，查询所有的列表，调用inv将列表发给钱包节点

七、钱包节点遍历这个列表，如果区块hash 不在自己的节点数据里，则将区块hash一个个发送给主节点，请求block数据data

八、主节点收到这个hash后，查询对应的block数据，并推送给钱包节点

九、钱包节点收到block数据后，添加到自己的区块和区块链中，继续余下的区块hash

十、矿工节点流程也一样

8. 钱包节点转账、主节点、矿工节点之间的完整交互逻辑

一、准备3个终端，各自设置NODE\_ID(3000,3001,3002)

二、在主节点中创建一个钱包,创建一个创世区块，将blockchain\_3000.db备份到文件blockchain\_genesis.db(给后面的钱包节点和挖矿节点作为初始区块备份用),并查看主节点的钱包余额是否为10

三、在钱包节点中主设置NODE\_ID=3001,拷贝blockchain\_genesis.db为blockchain\_3001.db,创建4个钱包地址,并查看各自的余额是否为0

四、在主节点中，给钱包中的前2个地址分别转帐8,6个 带是立即挖矿-mine标识，查看主节点中的余额是16,2个钱包节点的余额是8,6，切换到钱包节点查询4个钱包余额仍是0，创世钱包余额为10

五、启动主节点，钱包节点，看是否进行数据同步，此时钱包节点用应该会接收到节点数据(发现数据不对，要考虑utxoreset，需要重置余额表数据才能准确)

六、注意事项:当钱包节点更新区块时，要更新数据里的"L"字段对应的最新hash值，并更新blockchain中的tip字段值，保存l和tip是最新的hash值

七、启动主节点，等待接收钱包节点或矿工节点的请求数据

八、启动钱包节点，并将bc=GetObjectBlockchain(3001),sendVersion("localhost:3000",bc)消息给主节点,里面的数据是：(包含12个字节长度+序列化(Version,locaalhost:3001,1))

九、主节点收到钱包节点发过来的消息后，通过handleVersion反序列化，分析 command，进入相应的分支处理,通过高度进行比较。如果如果高度小于钱包节点，则继续调用 sendVersion接口，主节点高度大于钱包节点，则sendGetBlocks(3000),否则sendVersion(3001,bc)、

9. 怎么理解libp2p实现节点数据同步

一、终端1向本地的区块链添加了一个新的区块Block

二、终端1向终端2广播该信息

三、终端2将新的区块链跟本地的对比，发现终端1的更长，因此使用新的区块链替代了本地的区块链，然后将新的区块链广播给终端3

四、同上，终端3也进行更新

五、所有的3个终端节点都把区块链更新到了最新版本，同时没有使用任何外部的中心化服务

10. 运行Otto，编写一个简单的合约，将合约提交到虚拟机进行编译运行，附属上相关截图

1,到https://github.com/robertkrimen/otto下载最新版本otto

2,在gopath的src下创建github.com文件夹，在github.com文件夹下创建robertkrimen文件夹，并将下载的otto-master拷贝到此文件夹，并讲otto-master更名为otto

3,通过终端进入gopath的src路径下,然后下载gokpg.in/sourcemap.v1依赖包

3.1,go get gopkg.in/sourcemap.v1

4，创建工程testotto，与github.com、gokpg.in并列在go环境中的src文件夹下，

5，下面可以测试了，在main.go文件中，Run something in the VM，代码如下：

package main

import (

"github.com/robertkrimen/otto"

"fmt"

"net/http"

"log"

)

func main() {

//测试虚拟机能否运行js

http.HandleFunc("/", sayHttp)

err := http.ListenAndServe(":7777", nil)

if err != nil {

log.Fatal("ListenAndServe: ", err)

}

}

//parse js function

func parseJSFunc(w http.ResponseWriter,r \*http.Request) {

r.ParseForm()

for k, v := range r.Form {

if k =="jstest" {

//创建虚拟机

vm := otto.New()

var src=string(v[0])

//设置Web中传递的js代码，并在虚拟机运行js

vm.Run(src)

//调用js对象

var obj,\_=vm.Object("object1")

//设置js中的成员变量

obj.Set("age",1000)

//调用js成员成员变量

var age,\_=obj.Get("age")

var name,\_=obj.Get("name")

fmt.Println("name:",name,"age:",age)

//调用对象中的无参数方法

var meth,\_=obj.Call("sayHi",nil,nil)

fmt.Println(meth)

//调用对象中有参数的方法

var mpar,\_=obj.Call("sayHello",11,22)

fmt.Println(mpar)

}

}

fmt.Fprintf(w, "Hello world!")

}

func sayHttp(w http.ResponseWriter,r \*http.Request){

parseJSFunc(w,r)

}

即可实现go语言对Js的编译

function Person()

{

this.name = "kongyixueyuan";

this.age = 24;

this.sayHi = function(){

return this.age;

}

this.sayHello = function(a,b){

return a+b;

}

}

var object1 = new Person();

function Person() {  
  
 this.name = "颜学海"  
 this.age = 35  
 this.sayHi = function() {  
 Return "你好，我叫"+this.name+",我今年"+this.age+"岁了";  
 }  
}  
  
Var object = new Person();

